High-dimensional Graphical Model Search with **gRapHD R** Package

Gabriel C. G. de Abreu*,† David Edwards*,‡ Rodrigo Labouriau*,§

Abstract

This paper presents the **R** package **gRapHD** for efficient selection of high-dimensional undirected graphical models. The package provides tools for selecting trees, forests, and decomposable models minimizing information criteria such as AIC or BIC, and for displaying the independence graphs of the models. It has also some useful tools for analysing graphical structures. It supports the use of discrete, continuous, or both types of variables.

1 Introduction

We describe here the **R** package **gRapHD** which is designed to work with graphical models involving a large number of variables. These may be useful for modelling high dimensional networks in an wide range of biological applications (*e.g.* ecology [10]; gene expression [13]; proteomics [22]). Other applications are in computer sciences (*e.g.* internet [9]), engineering (*e.g.* complex electronic circuits), physics [8], sociology [17].

The graphical models used here are classes of multivariate distributions whose conditional independence properties are encoded by a graph in the following way. The random variables are represented as vertices (nodes), and two vertices are connected by an edge (line) when the corresponding variables are not conditionally independent given the other variables represented in the graph. Thus the absence of an edge connecting two vertices indicates conditional independence of the two corresponding variables given the other variables.

This type of model has been used in high dimensional contingency tables (*e.g.* log-linear models [5,16]), continuous variables (*e.g.* the covariance selection model described by [7]), and models containing both continuous and discrete variables [18]. Modern accounts of graphical models can be found in [11], [19], and [21].

*Department of Genetics and Biotechnology, Faculty of Agricultural Sciences, Aarhus University, DK-8830 Tjele, Denmark.

†E-mail: Gabriel.Abreu@agrsci.dk

‡E-mail: David.Edwards@agrsci.dk

§E-mail: Rodrigo.Labouriau@agrsci.dk

The use of graphical models for large numbers of variables can be very challenging, both because of computational limitations, and because of intrinsic statistical difficulties (particularly when the sample sizes are small). Consequently the use of such models has often been restricted to small problems. The package **gRapHD** presented here is intended for high-dimensional graphical modelling. The central functions, `minForest` and `stepw`, search respectively for the optimal forest and the optimal decomposable model, where optimality is typically defined in termed of an information criterion (AIC or BIC).

The the **gRapHD** package is here presented using three distinct examples, which are described in Section 2. The basic definitions and notations used throughout the paper are found in Section 3, and the structure of the `gRapHD` object class defined by the package is described in Section 4. Sections 5 to 7 present the functions in the package, using the examples previously introduced.

2 Three examples

The features of the **gRapHD** package will be presented using three different examples covering continuous, discrete, or both types of variables (examples 1, 2, and 3, respectively). We here describe the data and show the graphical model selected.

2.1 Example 1 - continuous: Periodontitis data

The data arise from a functional genomics study in gingival tissue [6]. The study investigates the differences in the gene expression profiles of interproximal papillae tissues of diseased (periodontitis) and health sites. Ninety healthy non-smoker patients with moderate to advanced periodontitis were used in this study. Each patient contributed with at least two diseased samples and a healthy papilla, if available. A total of 247 samples were collected (64 from healthy sites and 183 from diseased sites). The transcription profiles of the samples were evaluated using Affymetrix human genome arrays with 54,675 probe sets.

Only 64 arrays from independent diseased sites were considered. The data were pre-processed using `justRMA` from the **affy** package [15]. To reduce the amount of data, probes with variance < 0.62 were omitted from the analysis. Thus our dataset is composed of 1,545 probes, from 64 different patients. The objective is to characterize the gene co-expression network in patients with periodontitis. The model selected by the `stepw` function can be seen in Figure 1 (A).

2.2 Example 2 - discrete: HapMap data

The goal of the International HapMap Project is to characterize human genetic variation [20]. The project recorded differences in the sequence of bases that composes the DNA,

the *SNPs* (*S*ingle *N*ucleotide *P*olymorphisms). There are four different bases (A, T, G, and C) that can occur at each position in the DNA (*locus*), and if different individuals have different bases at a locus, the locus can be considered polymorphic (provided that none of the alleles are too rare). Each different base occurring in one locus is considered as an allele.

We use here only the polymorphic SNPs from the Yoruba population (Ibadan, Nigeria - West Africa) with complete data. From this population, we selected the chromosome 17, representing 606 SNPs in 176 individuals, for synonymous coding SNPs (SNPs which even with different structure code the same amino acid) with minor allele frequency greater than 0.25 (download on the 3^{rd} June 2009 from the HapMart website - `www.hapmap.org`). After eliminating 9 individuals with very high missing values percentage we obtained 334 loci without missing values. Using the information of the stated reference allele we codified the genotypes of the individuals as: "0" for homozygous wild type (individual with the reference allele in both DNA strands); "1" for heterozygous (individual with only one copy of the reference allele); and "2" for homozygous mutant (individual with no copy of the reference allele). The objective is to determine the relationship between different loci through the network structure. The optimal decomposable representation of the network (minimum BIC) is displayed in Figure 1 (B).

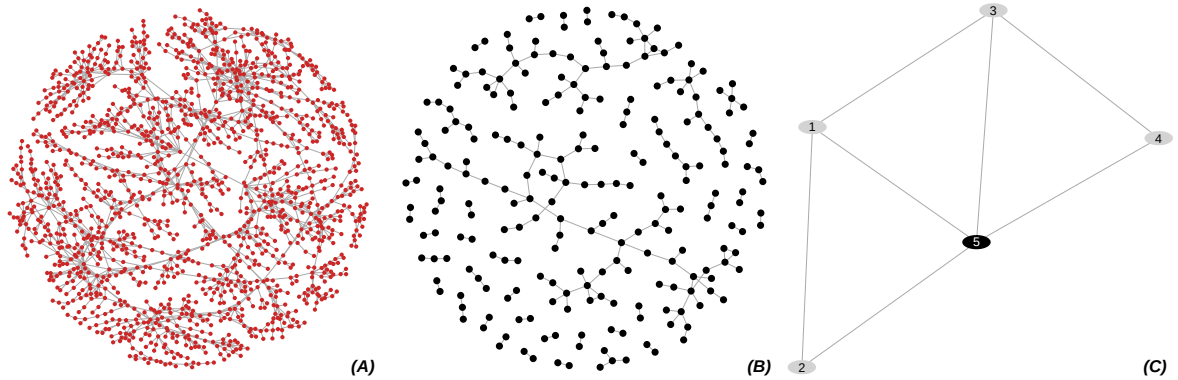

Figure 1: **Three inferred networks**. (A) Periodontitis data (example 1); (B) HapMap data (example 2); (C) Iris data (example 3).

2.3 Example 3 - mixed: Iris data

The iris flower data were introduced by Anderson in 1935, but is also known as the Fisher's iris data (1936). A sample of 150 plants, 50 from each of three species (*Iris setosa*, *Iris versicolor*, and *Iris virginica*), had the sepal and petal lengths and widths recorded. The objective is to describe the structure of correlation between the different

measurements considering the 3 different species. The final model can be seen in Figure 1 (C).

3 Basic definitions and notation

We here give a brief sketch of the theory of graphical models. For a more complete account see for example [19]. Graphical models combine graph theory and probability theory. Each vertex represents a random variable, and two vertices are connected when they are not conditionally independent given the remaining variables. For example, from the graph presented in Figure 2 (A), we see that the variables 1 and 2 are not conditionally independent given variables 3, 4 and 5, but that variables 1 and 4 are conditionally independent given the variables 2, 3 and 5.

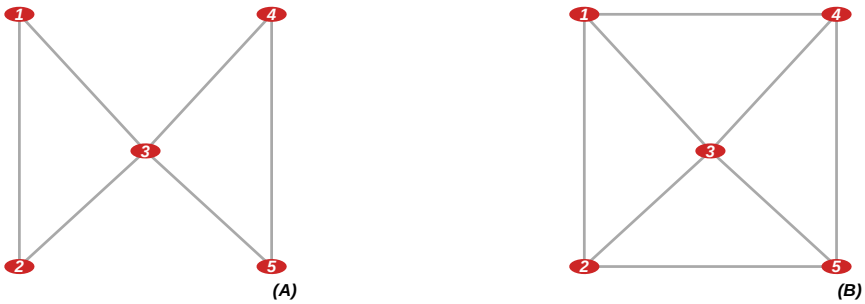

Figure 2: **Example of triangulation**. (A) The structural relationship between the 5 vertices indicates that vertices 1 and 2 are conditionally independent of vertices 4 and 5, given vertex 3. (B) Adding the edges (1,4) and (2,5) renders the graph non-triangulated.

Define a graphical model as an undirected graph $\mathbf{G}$=($\mathbf{V}$,$\mathbf{E}$), where $\mathbf{V} = \{v_1, ..., v_p\}$ is the set of p vertices (p finite), and $\mathbf{E}$ is the set of edges, a subset of $\mathbf{V} \times \mathbf{V}$ (unordered pairs), where multiple edges and self-loops are not allowed. Furthermore, an edge $e = (u, v) \in \mathbf{E}$ indicates that the variables associated to u and v are not conditionally independent given all the other variables. If we are only interested in a subset of such relations, we can define a subgraph of $\mathbf{G}$ as $\mathbf{G}_A = (\mathbf{A}, \mathbf{E}_A)$, where $\mathbf{A} \subseteq \mathbf{V}$ and $\mathbf{E}_A \subseteq \mathbf{E}$ [1]. For example, in Figure 2 (A), $\mathbf{V} = \{1, 2, 3, 4, 5\}$ and $\mathbf{E} = \{(1, 2), (1, 3), (2, 3), (3, 4), (3, 5), (4, 5)\}$, and $\mathbf{A} = \{1, 2, 3\}$ renders the subgraph to the graph $\mathbf{E}_A = \{(1, 2), (1, 3), (2, 3)\}$.

A graph is complete if every pair of vertices is connected by an edge. If a subgraph is maximally complete, it is called a *clique*: in this case the addition of any other vertex would renders the subgraph incomplete. In Figure 2 (A), $\{1, 2, 3\}$ and $\{3, 4, 5\}$ are cliques.

In a graph $\mathbf{G}$, two vertices, u and v, are said to be connected if there is a sequence $u = v_1, ..., v_k = v$ of distinct vertices such that $(v_{i-1}, v_i) \in \mathbf{E}$, $\forall\, i = 2, ..., k$. The sequence $u = v_1, ..., v_k = v$ is called *path*. In the Figure 2 (A), the vertices 1 and 5 are connected since there exists a path between them, for example $1, 2, 3, 5$. A *cycle* is a path which the end vertices are the same ($u = v$), as $1, 2, 3, 1$ in Figure 2 (A). A cycle $u = v_1, ..., v_k = u$ is *chordless* if v_i and v_j are only connected by an edge when $|i-j| = 1$. A graph is called *triangulated* if it has no chordless cycles of length greater than three. For example, the graph in Figure 2 (A) is triangulated, but the graph in Figure 2 (B) is not, since the cycle $1, 2, 5, 4, 1$ of length four is chordless.

A subset $\mathbf{C} \subseteq \mathbf{V}$ separates two disjoint subsets of $\mathbf{V}$, $\mathbf{A}$, and $\mathbf{B}$, if all paths from $v \in \mathbf{A}$ to $u \in \mathbf{B}$ pass through $\mathbf{C}$. In addition, a triple $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ of disjoint subsets of $\mathbf{V}$ decomposes the graph $\mathbf{G}$=($\mathbf{V}$,$\mathbf{E}$) if: (1) $\mathbf{V} = \mathbf{A} \cup \mathbf{B} \cup \mathbf{C}$, (2) $\mathbf{C}$ separates $\mathbf{A}$ from $\mathbf{B}$, and (3) $\mathbf{C}$ is complete. This definition implies that a graph is *decomposable* if it is complete, or exists a decomposition $(\mathbf{A}, \mathbf{B}, \mathbf{C})$, with $\mathbf{A} \neq \emptyset$ and $\mathbf{B} \neq \emptyset$, into decomposable subgraphs $\mathbf{G}_{A\cup C}$ and $\mathbf{G}_{B\cup C}$. A graph $\mathbf{G}$=($\mathbf{V}$,$\mathbf{E}$) is decomposable if and only if it is triangulated [19]. The example shown in Figure 2 (A) is decomposable, with $A = \{1, 2\}$, $B = \{4, 5\}$, and $C = \{3\}$.

The cliques $C_1, ..., C_k$ in a triangulated graph can be numbered in such a way that for all $j = 1, ..., k$, $H_j = C_i \cup ... \cup C_j$, $R_j = C_j \backslash H_{j-1}$, and $S_j = H_{j-1} \cap C_j$ gives that (1) for all $i > 1$ there is a $j < i$ such that $S_i \subseteq C_j$, and (2) the sets S_i are complete for all i [19]. This sequence of cliques is called a *perfect sequence*. The sets H_j, R_j, and S_j are named histories, residuals, and separators, respectively. A perfect sequence of the graph in Figure 2 (A) is $\{2, 3, 1, 4, 5\}$, with cliques $C_1 = \{1, 2, 3\}$ and $C_2 = \{3, 4, 5\}$, respective separators $S_1 = \emptyset$ and $S_2 = \{3\}$, histories $H_1 = \{1, 2, 3\}$ and $H_2 = \{1, 2, 3, 4, 5\}$, and residuals $R_1 = \{1, 2, 3\}$ and $R_2 = \{4, 5\}$.

A key property of decomposable graphs is that the probability densities of such models can be factorized as:

$$f(\mathbf{v}) = \frac{\prod_{C\in\mathcal{C}} f(\mathbf{v}_C)}{\prod_{S\in\mathcal{S}} f(\mathbf{v}_S)^{\nu(S)}},$$

where $\mathcal{C}$ is the class of cliques in a perfect sequence, and $\nu(\mathbf{S})$ is the number of times that $\mathbf{S}$ occurs as a separator in this perfect sequence (possibly including the empty set) [19]. Furthermore, these models have an explicit formula for the maximum likelihood estimators.

A *forest* is a graph containing no cycles. It may be composed of several connected components called trees, i.e. a tree is a connected acyclic graph [2]. Given a set of edge weights, a *minimum spanning forest* $\mathbf{G}_m = (\mathbf{V}_m, \mathbf{E}_m)$ of a graph $\mathbf{G}$=($\mathbf{V}$,$\mathbf{E}$) is a forest with $\mathbf{V}_m = \mathbf{V}$ and $\mathbf{E}_m \subset \mathbf{E}$ that has minimum sum of edge weights, among all possible such forests. Examples of edge weights are the contribution of each edge to the BIC or the contribution to minus the log-likelihood. Figure 3 (A) shows a graph with four connected components, and Figure 3 (B) shows a spanning forest of this graph.

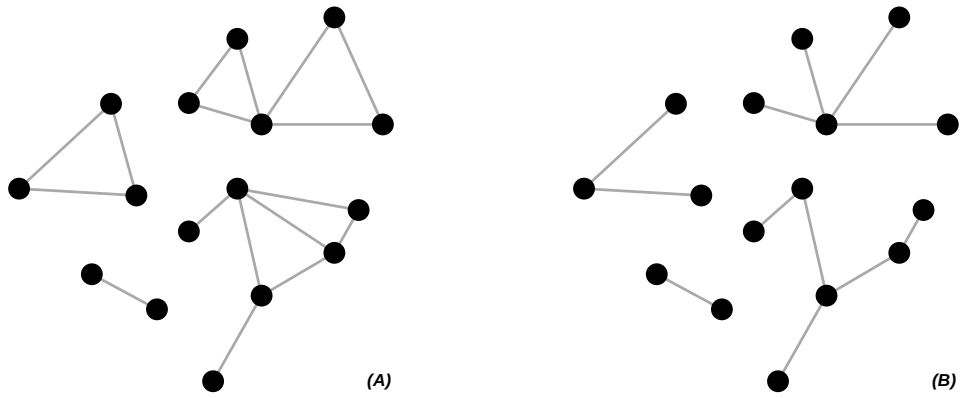

Figure 3: **Example tree and forest**. (A) Graph with four connected components. (B) One possible spanning forest (each connected component is a tree) for the graph in (A).

The variables in the model can be discrete, continuous, or both (mixed). In the first case, in which each variable assumes a value in a set of levels, the models are based on the multinomial distribution. In the second case, the models are based on the multivariate Gaussian distribution. In the mixed case the CG (conditional Gaussian) distribution is assumed; the variances can be homogeneous or heterogeneous across different levels of the discrete variables. Furthermore, a mixed model is strongly decomposable when its graph is triangulated and no forbidden paths occur. A *forbidden path* is a path between two non-adjacent discrete vertices passing through only continuous vertices, as showed in Figure 4. For more details see [19] pages 7-12.

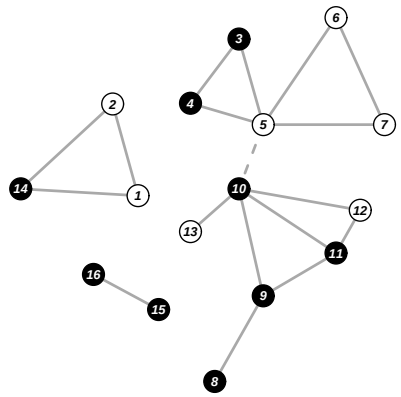

Figure 4: **Forbidden path**. Continuous variables are represented as circles, and discrete variables as dots. The edge (5,10) creates a forbidden path in the graph, since a continuous variable (variable 5) connects non-adjacent discrete variables (variables 3 and 4 to variables 8, 9, 10, and 11).

In this paper we describe the main features of the new **R** package **gRapHD**, and how

it can be used for graphical model search. Most of the core functions are programmed in **ANSI C**, due to its computational efficiency. The package as well as the code are available from `http://CRAN.R-project.org/`.

4 The gRapHD object

The **gRapHD** package groups a number of functions designed for efficient selection of high-dimensional undirected graphical models. The set of variables can contain only discrete, continuous, or both types. All the information about the model is stored in a `gRapHD` object which contains the following elements:

- `edges`: Matrix with 2 columns, each row representing one edge, and each column one of the vertices in the edge.
- `p`: Number of variables (vertices) in the model.
- `stat.minForest`: Measure used (LR, AIC, BIC, or a user defined function) by the `minForest` function.
- `stat.stepw`: Measure used (LR, AIC, BIC, or a user defined function) by the `stepw` function.
- `statSeq`: Numeric vector with the value of the `stat.minForest` for each edge found by `minForest`, or the change in the `stat.stepw` for each edge found by `stepw`.
- `vertNames`: Vector with the original vertices' names. If no names are attributed, then the vertices will be named according to their original order in the data.
- `numCat`: Vector with number of levels for each variable (0 if continuous).
- `homog`: `TRUE` if the covariance is homogeneous (only used in the mixed case).
- `numP`: Vector with the number of estimated parameters for each edge.
- `minForest`: Vector of length 2, with the row indexes of the first and last edges found by the `minForest` function.
- `stepw`: Vector of length 2, with the row indexes of the first and last edges found by the `stepw` function.

The `gRapHD` object, besides being the result of a model search, functions as an input parameter in most other functions in the package. A null model can be created using the function `as.gRapHD`, as in

```
R> m <- as.gRapHD(NULL)
List of 9
 $ edges    : int[0 , 1:2]
 $ p        : int 0
 $ stat.user: chr "LR"
 $ statSeq  : num(0)
 $ numCat   : int(0)
 $ homog    : logi TRUE
 $ numP     : num(0)
 $ vertNames: logi NA
 $ userDef  : num [1:2] 0 0
 - attr(*, "class")= chr "gRapHD"
```

5 Search for graphical models

The core functions in the package are `minForest` and `stepw`. The `minForest` function finds a minimal AIC or BIC forest, or the maximum likelihood tree for the data. The `stepw` function performs forward search for the triangulated graph that minimises a given measure. The minimized measure used can be either `-LR` (likelihood-ratio), `BIC` (Bayesian Information Criteria), `AIC` (Akaike Information Criteria), or a function specified by the user.

All calculations use the data set specified by the `dataset` parameter, which holds the raw data, in `dataframe` format, with the column storage mode defining its type (discrete or continuous). The discrete variables are defined as factors, and the continuous as numerical. In the first case, there should be more than one level (otherwise there is no gain in using such variable), and all levels must be represented in the sample. If all variables are continuous, the dataset can be given as a numeric matrix. Missing values (`NA`) are not allowed. The variables are referred by their indexes in the `vertNames` attribute. In this way, the edge representation in the `gRapHD` object is in the format (v_1, v_2), where v_1 and v_2 are the indexes in `vertNames`, and $v_1 < v_2$. For example, the Iris data is presented as

```
R> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 ...
```

The discrete variable (`Species`) is identified as a factor with three levels. All other

variables are continuous (as indicated by the numerical type '`num`'). The columns names (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`, and `Species`) are stored in the `vertNames` attribute of the `gRapHD` object. The variables are always referred to as the original column number in the `dataset`, *e.g.* the variable `Sepal.Width` is variable 2, while `Species` is variable 5. The edge connecting `Sepal.Width` and `Species` is represented by a row in the `edges` attribute consisting of the vector `c(2,5)`.

5.1 Search for minimum spanning forests

The function `minForest` searches for an optimal tree or forest using the algorithm of [3]. If the ML measure is used, the function returns a tree, but if the AIC or BIC is used, the function may return a forest or a tree. Per default the BIC measure is used. Starting from an empty edge set, the algorithm repeatedly adds the edge that optimizes the selected measure. Only edges that preserve the tree/forest structure can be added, i.e., no cycles can be generated. The procedure continues until no more edges can be added. For example, if the selected measure is the BIC, at first the pairwise values are computed and in each step the edge that reduces the most the total BIC is added, if it exists. The algorithm is

Step 1: Calculate the BIC for all possible edges.

Step 2: Select the edge that improves the most the model's BIC.

Step 3: If there is no such edge, stop.

Step 4: Test if the addition of this edge creates a cycle or a forbidden path.

Step 5: If it does, select the next edge with best improvement and return to **Step 3**.

Step 6: Add the edge to **E**, remove it from the list of possible edges, and return to **Step 2**.

For mixed models, the algorithm finds the strongly decomposable forest that minimizes the selected measure [12].

For the periodontitis data, the summary of the minimum spanning forest is presented below. The graph found is actually a tree and not a forest, since all vertices are connected ($p - 1$ edges). All variables are continuous, and the measure minimised was the BIC (default).

```
R> library("gRapHD")
R> periodontitisForest <- minForest(periodontitisData)
gRapHD object
Number of edges       = 1544
```

```
Number of vertices    = 1545
Model                 = continuous
Statistic (minForest) = BIC
Edges from minForest  = 1...1544
```

5.2 Forward search

The function `stepw` searches for decomposable models minimising a given measure by adding edges to a initial model, also decomposable. The algorithm is also iterative, determining at each step the add-eligible edges, i.e., the edges that if added preserve the triangularity. Among these edges, the one that reduces the most the selected measure is added to the graph. The algorithm, showed below, stops when no more add-eligible edges are found. The structure of components in the starting model is preserved as default, which means that if the function starts from a forest with k isolated components, the final model will also have k isolated components. This can be changed setting the option `join` to `TRUE`.

Step 1: Calculate the BIC for all add-eligible edges.

Step 2: Select the edge that improves the most the model's BIC.

Step 3: If there is no such edge, stop.

Step 4: Add the edge to **E**, and return to **Step 1**.

For example, if the function is applied on the tree found in the previous section, 999 iterations are necessary (998 edges are added to the tree).

```
R> periodontitisForward <- stepw(periodontitisForest,periodontitisData)
gRapHD object
Number of edges       = 2542
Number of vertices    = 1545
Model                 = continuous
Statistic (minForest) = BIC
Statistic (stepw)     = BIC
Edges from minForest  = 1...1544
Edges from stepw      = 1545...2542
```

The default measure is the BIC. We can see that in the final model, the first 1,544 edges were found by `minForest` (edges 1 to 1,544), and the last 998 by `stepw` (edges 1,545 to 2,542). The final graph can be seem in Figure 1 (A).

The algorithm can start from an empty model (which is always decomposable). If we use the iris data, the final model (Figure 1 C) found is the same as if it had started

from the forest/tree graph. The code for it is showed below. Initially a `gRapHD` object is generated, from an empty model with 5 variables, being the first 4 continuous and the last discrete with 3 levels. The model is considered heterogeneous. As we are starting from 5 isolated components, they are allowed to be joined.

```
R> irisEmpty <- as.gRapHD(matrix(integer(0),,2),p=5,numCat=c(0,0,0,0,3),
+                         homog=FALSE)
R> irisForward <- stepw(irisEmpty,iris,join=TRUE)
```

5.3 Computational performance

The performance of **gRapHD** was evaluated using a Intel(R) Xeon(R) CPU E5450 3.00GHz with 31Gb of RAM, running Linux 64 bits. The memory use and CPU time of the three examples are presented in Table 1.

Example	Number of	Number of edges added		CPU time (sec.)		Memory (Mb)	
	vertices	`minForest`	`stepw`	`minForest`	`stepw`	`minForest`	`stepw`
Periodontitis	1,545	1,544	998	9.27	2024.14	36.64	0.0956
HapMap	334	189	5	0.18	2.18	1.66	0.01
Iris	5	4	3	0.001	0.003	0.002	0.0003

Table 1: **Computational performance**. CPU time (for the whole procedure) and memory use of the three examples.

We see that the `minForest` function is highly efficient in terms of CPU time but may require substantial memory.

Additionally, we performed a simulation study where the functions `minForest` and `stepw` were evaluated. Data sets from multivariate normal distributions were generated with 10 to 5,000 variables (at intervals of 20). Each of these data sets were used to infer the minimum spanning forest and a more complex decomposable graph starting from this forest. The computational performance is showed in Figure 5. As the `stepw` function depends on the final complexity of the model (number of edges added to the model), the CPU time shown is for one (average) iteration only. The memory demand for the `minForest` function grows quadratically in the number of vertices, while the memory demand for the `stepw` function grows linearly in the number of vertices. The CPU time also grows quadratically in both functions.

In conclusion, the `minForest` function runs much faster than the `stepw`, allowing the selection of a skeleton of the model in a forest-like structure (sparse) in a very short time. We remark that it appears to be much more efficient to start the `stepw` function from the forest found by `minForest`, rather than from the null model.

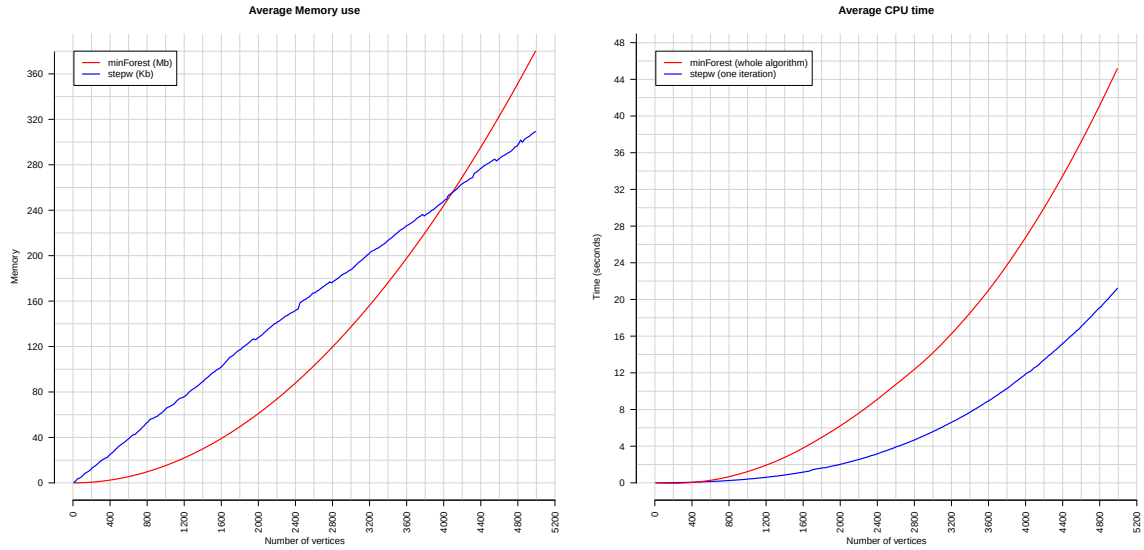

Figure 5: **Computational performance**. Memory use and CPU time. The curve for the CPU time of the `stepw` function reflects an average iteration, while for the `minForest` function it is for the whole algorithm.

6 Analysis of the graphical structure

When high-dimensional models are studied, plotting the model found is probably not the best way to visualise the result, as can be seen in Figure 1. The plotting algorithm used here is time consuming, and may need a large number of iterations to generate a graph with no edges crossing each other. Therefore different ways to analyse the resulting model have to be used. The package contains a number of functions which could be useful for this task:

- `adjMat`: Returns the adjacency matrix based on a list of edges.
- `fit`: Calculate a model's -2×log-likelihood, AIC, and BIC.
- `Degree`: Returns the degree of a set of vertices.
- `DFS`: Finds all vertices reachable from one specific vertex (assuming that there are no cycles).
- `findEd`: Finds all add-eligible edges to a given triangulated graph, preserving triangularity. In the case of a mixed graph, forbidden edges are not returned.
- `neighbours`: Returns all vertices with a direct connection with a vertex `v`.

- `MCS`: Returns a perfect numbering of the edges.
- `modelDim`: Calculates the number of free parameters corresponding to each edge.
- `modelFormula`: Finds the model's formula [19, pages 213-216].
- `neighbourhood`: Finds the set of vertices with up to a given distance from a given vertex.
- `perfSets`: Finds a perfect sequence, returning the cliques, histories, residuals, and separators of a given triangulated graph.
- `shortPath`: Calculates the shortest path between a vertex `v` and all other vertices.
- `SubGraph`: Based on a list of vertices, generates a subgraph.
- `summary`: Gives details about the model's structure.
- `jTree`: Finds a junction tree of a graph.
- `CI.test`: Calculates the deviance and adjusted degrees of freedom for the conditional independence test.

The degree of the periodontitis model shows two vertices with high number of direct neighbours (degree 20). These vertices could be "hubs", with important action in the network. We can also see that there is a high number of "leafs" in the graph, as 382 (24.72%) vertices present only one edge.

```
R> table(Degree(periodontitisForward))
  1   2   3   4   5  6  7  8  9 10 11 12 13 14 15 16 17 20
382 379 249 185 116 83 53 28 18 17 13  6  5  6  1  1  1  2
```

We can zoom in the neighbourhood (up to the second neighbour, for example) of these two "hubs".

```
R> vertices <- which(Degree(periodontitisForward)==20)
R> neigh670 <- neighbourhood(periodontitisForward,orig=vertices[1],rad=2)
R> pos <- plot(periodontitisForward,numIter=4000,vert.labels=FALSE,
+              vert.radii=.006,vert.hl=vertices,col.hl=c("red","blue"))
R> plot(periodontitisForward,vert=neigh670$v[,1],numIter=1000,
+       vert.hl=vertices[1],vert.radii=.013,col.hl="red",cex.vert.label=.4)
R> plot(periodontitisForward,vert=neigh1123$v[,1],numIter=1000,
+       vert.hl=vertices[2],vert.radii=.013,col.hl="blue",cex.vert.label=.4)
```

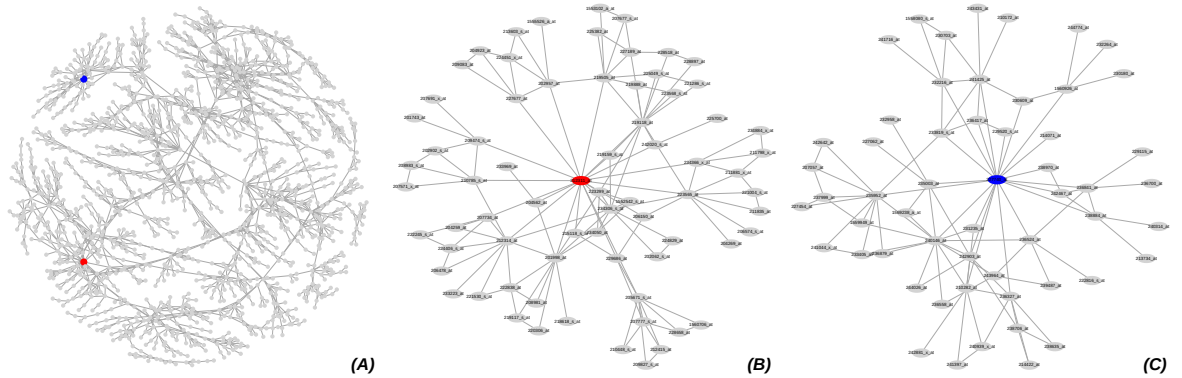

Figure 6: **Zooming in specific regions of a large graph**. (A) The final graph of the periodontitis data, highlighting vertices 670 (red) and 1,123 (blue). (B) The neighbourhood of vertex 670, including only vertices within a radius of two. (C) The neighbourhood of vertex 1,123, including only vertices within a radius of two.

From Figure 6 we see that restricting the plot to a smaller neighbourhood allows details to become more visible. The same plot could also be produced using the functions `neighbours` and `SubGraph`.

For the HapMap data, the degree shows that we actually have a high number of isolated vertices (104, 31.14%). The general structure of the model is very close to a forest, as only 5 edges are added by the `stepw` function. The vertices with highest degree have 5 edges.

```
R> table(Degree(HapMapForward))
  0   1   2   3   4   5
104 133  58  22  12   5
```

The log-likelihood, BIC, and AIC of a model can be obtained using the function `fit`. For example, if we are interested in test in a mixed model if the structure of covariance is homogeneous or heterogeneous, we could use the this information.

```
R> fit(edges=irisForward,dataset=iris,homog=FALSE)
Number of parameters   -2*Log-likelihood        AIC        BIC
             39.0000            410.1435   488.1435   605.5582
R> fit(edges=irisForward,dataset=iris,homog=TRUE)
Number of parameters   -2*Log-likelihood        AIC        BIC
             22.0000            551.3188   595.3188   661.5528
```

The `perfSets` function finds the cliques structure of the graph, also returning the lists of separators, histories, and residuals, as described in [19] (pages 14-15). If the graph is not (strongly) decomposable, the function returns the value zero. For the periodontitis data we have

```
R> sets <- perfSets(periodontitisForward)
# only the first two elements of each are shown here
List of 4
    $ cliques :List of 1395
     ..$ : int [1:2] 1 335
     ..$ : int [1:3] 394 335 474
    $ histories :List of 1395
     ..$ : int [1:2] 1 335
     ..$ : int [1:4] 1 335 394 474
    $ separators :List of 1395
     ..$ : NULL
     ..$ : int 335
    $ residuals :List of 1395
     ..$ : int [1:2] 1 335
     ..$ : int [1:2] 394 474
```

The `shortPath` function returns the shortest path length between vertices in the graph (considering that each edge has length one). If two vertices are not connected, i.e., there is no path between them, it is returned `Inf`. In the periodontitis data, the vertex 670 has a direct connection with 20 other vertices, and the most far vertex has a distance of 26 from it, while the graph's diameter (longest shortest path) is 46. Note that vertex 670 has a distance of 0 to itself.

```
R> table(shortPath(periodontitisForward,v=670))
 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 1  20  50  58  72  78  84  77  80  97 134 130 126 116  98  91  80  70
18  19  20  21  22  23  24  25  26
48  15   4   4   4   3   3   1   1
```

7 Plotting a graphical model

Some examples of plots generated by the `plot.gRapHD` function have been given above. The function uses S3 method for the class `gRapHD`, so the regular `plot` function can be used. As default, when a `gRapHD` object is plotted, all discrete variables are pictured as black circles, and the continuous variables as grey circles. But the function is flexible, and the user can define different colours, shapes, and sizes for each vertex. The [14] algorithm is used to place the vertices in the plotting area [4]. The algorithm is iterative and uses attractive and repulsive forces for placing the vertices. This technique is time consuming and cannot guarantee a clear plot, as the one in Figure 1 (A), which used 4,000 iterations, a number not sufficient to untangle it.

The more complex a graph is, the more difficult is its visualisation. For this reason, the `plot.gRapHD` function has a number of optional parameters that allow the user to manipulate the appearance of the plot. It is possible to plot only the edges, or not to label the vertices; to highlight some vertices with different colours, shapes, and/or sizes. For example we could show in the graph where the neighbourhood of the vertices 670 and 1,123 are, for the periodontitis data, as shown in Figure 7.

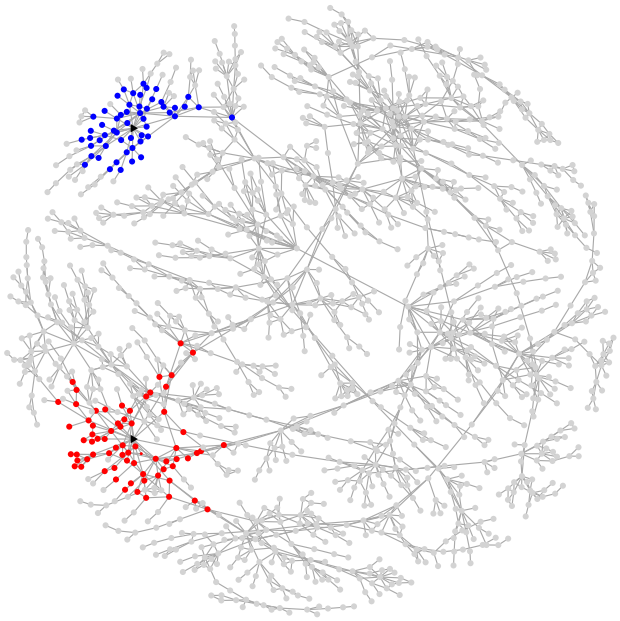

Figure 7: **Highlighting specific regions in a complex graph**. The vertices 670 and 1,123 are pictured in black as triangles, while the respective neighbourhoods are picture in red and blue.

```
R> hlv <- c(670,1123,neigh670$v[-1,1],neigh1123$v[-1,1])
R> hlc <- rep(c("black","red","blue"),c(2,length(neigh670$v[-1,1]),
+             length(neigh1123$v[-1,1])))
R> vs <- rep(0.005,periodontitisForward$p)
R> vs[c(neigh670$v[,1],neigh1123$v[,1])] <- .008
R> vs[c(670,1123)] <- .01
R> sb <- rep(1,periodontitisForward$p)
R> sb[c(670,1123)] <- 3
R> vs[901]<-0
R> plot(periodontitisForward,coord=pos,numIter=0,vert.hl=hlv,col.hl=hlc,
+       vert.labels=FALSE,vert.radii=vs,symbol.vert=sb)
```

The code below can be used to identify isolated components in a graph. The largest components in the HapMap data are displayed in Figure 8.

```
R> sp <- shortPath(HapMapForward)
R> comp <- rep(0,HapMapForward$p)
R> i <- 0
R> while(length(which(comp==0))>0) {
R>   i <- i + 1
R>   if (comp[i] == 0) {
R>     ind <- which(sp[i,]<HapMapForward$p) #finite
R>     comp[ind] <- max(comp) + 1
R>   }
R> }
R> v <- c(which(comp==18),which(comp==21),which(comp==15),
          which(comp==1),which(comp==9))
R> col <- rep(colours()[c(133,124,258,150)],c(14,13,39,79))
R> plot(HapMapForward,vert=v,lwd.ed=3,vert.hl=v,symbol.vert=rep(0,length(v)),
+   vert.radii=rep(.012,length(v)),col.hl=col,numIter=3000,cex.vert.label=.7)
```

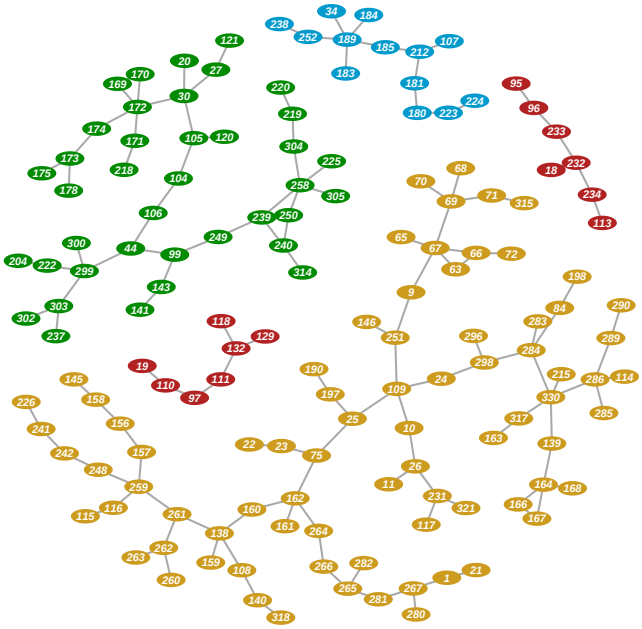

Figure 8: **Largest components in the HapMap graph**. The components with more than 5 vertices are plotted in different colours - red (7 vertices); blue (13 vertices); green (39 vertices); and yellow (79 vertices).

8 Concluding remarks

We have described an **R** package for efficient selection of high-dimensional undirected graphical models, with functions not available in other packages. The objective of the package is to provide an efficient way to search for models in the classes of forests and decomposable graphs (discrete, continuous, and mixed). The computational performance depends on the model complexity, where sparse models demand much less resources. It is easy to convert `gRapHD` objects into graph or model objects supported by other packages; it is also easy to extract information directly from them. There is a technical limitation of 65,000 variables, due to numerical representation.

Acknowledgements

Gabriel C. G. Abreu is financed by SABRETRAIN Project, funded by the Marie Curie Host Fellowships for Early Stage Research Training, as part of the 6^{th} Framework Programme of the European Commission. R.L. was partially supported by the project "Metabolic programming in Foetal Life", Danish Research Agency, Ministry of Science Technology and Innovation.

References

[1] Bela Bollobas. *Modern Graph Theory*. Springer, 2000.

[2] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[3] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.

[4] Gabor Csardi and Tamas Nepusz. The **igraph** software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[5] J. N. Darroch, S. L. Lauritzen, and T. P. Speed. Markov fields and log-linear interaction models for contingency tables. *Annals of Statistics*, 8:522–539, 1980.

[6] Ryan T. Demmer, Jan H. Behle, Dana L. Wolf, Martin Handfield, Moritz Kebschull, Romanita Celenti, Paul Pavlidis, and Panos N. Papapanou. Transcriptomes in healthy and diseased gingival tissues. *Journal of Periodontology*, 79(79):2112–2124, 2008.

[7] A.P. Dempster. Covariance selection. *Biometrics*, 29:157–175, 1972.

[8] S. Dhamodaran, A. Saad, and D. Fink. Application of network theory for the description of nanocluster distributions in ion track electronics. *Radiation Effects and Defects in Solids*, 163(9):749–759, September 2008.

[9] S.N. Dorogovtsev and J.F.F. Mendes. *Evolution of Networks: From Biological Networks to the Internet and WWW*. Oxford University Press, 2003.

[10] Jennifer A. Dunne, Richard J. Williams, and Neo D. Martinez. Food-web structure and network theory: The role of connectance and size. *Proceedings of the National Academy of Sciences*, 99(20):12917–12922, 2002.

[11] David Edwards. *Introduction to Graphical Modelling*. Springer-Verlag New York Inc., 2000.

[12] David Edwards, Gabriel Coelho Gonçalves Abreu, and Rodrigo Labouriau. Selecting high-dimensional mixed graphical models using minimal aic or bic forests. *BMC Bioinformatics*, 11(1):18, 2010.

[13] Jeremiah J Faith, Boris Hayete, Joshua T Thaden, Ilaria Mogno, Jamey Wierzbowski, Guillaume Cottarel, Simon Kasif, James J Collins, and Timothy S Gardner. Large-scale mapping and validation of escherichia coli transcriptional regulation from a compendium of expression profiles. *PLoS Biology*, 5(1):e8, 01 2007.

[14] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[15] Laurent Gautier, Leslie Cope, Benjamin M. Bolstad, and Rafael A. Irizarry. **affy**—analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004.

[16] Leo A. Goodman. The analysis of multidimensional contingency tables when some variables are posterior to others: A modified path analysis approach. *Biometrika*, 60(1):179–192, April 1973.

[17] J. Krause, D. P. Croft, and R. James. Social network theory in the behavioural sciences: Potential applications. *Behavioral Ecology and Sociobiology*, 62(1):15–27, November 2007.

[18] S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17(1):31–57, 1989.

[19] Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[20] The International HapMap Consortium. The international hapmap project. *Nature*, 426(6968):789–796, December 2003.

[21] Joe Whittaker. *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, 1990.

[22] Nir Yosef, Martin Kupiec, Eytan Ruppin, and Roded Sharan. A complex-centric view of protein network evolution. *Nucleic Acids Research*, 37(12):e88, July 2009.